

Imperial College London
Department of Earth Science and Engineering
MSc in Environmental Data Science and Machine Learning

Independent Research Project
Project Plan

Dynamic Fault-Tolerant Job Shop Scheduling using Ant Colony Optimisation

by

Benjamin Gorrie

Email: bg721@ic.ac.uk

GitHub username: [esemsc-bg721](https://github.com/esemsc-bg721)

Repository: <https://github.com/ese-ada-lovelace-2024/irp-bg721>

Supervisors:

Dr Marijan Beg

Dr Parastoo Salah

June 2025

Abstract

In this project plan, we aim to describe how Ant Colony Optimisation (ACO) can be applied to a Dynamic Job Shop Scheduling Problem (DJSSP), particularly one in which faults may occur. We outline why ACO is well-suited to this task and why this is an important problem to solve. We describe our plan for the rest of this project, and any challenges we may face along the way. Success will be defined by the ability of our code to adapt quickly and maintain feasible solutions under disruptions.

Contents

1	Introduction	1
2	Problem Description	2
2.1	Ant Colony Optimisation	2
2.2	Dynamic Job Shop Scheduling Problem	3
2.3	ACO applied to DJSSP	4
3	Significance	4
4	Review of Existing Work	4
5	Objectives	5
6	Future Plan	5
7	Risks	5
8	AI Acknowledgement Statement	5
	References	6

1 Introduction

Ant Colony Optimisation (ACO) is a technique which is inspired by how ants forage for food. ACO can be used to find “good” paths through graphs by simulating many digital agents (ants) and letting them traverse the graph, leaving behind trails of pheromones of different intensities depending on how good the final path is. As the ants are more attracted to paths with higher pheromone levels, this leads to ants converging on the path with the strongest pheromone trail, usually a near-optimal path.

ACO can be used to solve problems which can therefore be reduced to finding paths through graphs. There are many applications of this in practice (see [10]), with the most notable perhaps being vehicle routing and scheduling. These are not new problems, and there already exists extensive literature covering both of them. There are also excellent Python packages available for both of these applications [12] [7]. However, most of the existing literature covers static problems, where the edges and vertices of the graph we wish to find a path through are known in advance and do not change. Similarly, there does not appear to be a widely used Dynamic Job Shop Scheduling (scheduling where the set of jobs can change in real time) or dynamic routing Python package.

Thus, the aim of the project will be to create a Python package that addresses the Dynamic Job Shop Scheduling Problem (DJSSP) using ACO. As the pheromone trails that ants leave

behind can be reused, ACO should be a particularly efficient way of finding a new schedule as good connections between vertices do not need to be recalculated, and a new schedule does not need to be computed from scratch.

2 Problem Description

2.1 Ant Colony Optimisation

As previously described, ACO is a metaheuristic introduced by Marco Dorigo [1] used to find paths through graphs. Assuming we start with m ants, the k^{th} ant starts at a vertex i and moves to a previously unvisited vertex j at time t with probability

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in \text{allowed}_k} [\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta}$$

where we assume that $j \in \text{allowed}_k$, else the above quantity is 0. $\tau_{ij}(t)$ and η_{ij} are at the core of the algorithm and require a bit of explaining.

$\tau_{ij}(t)$ is the strength of pheromone deposited from vertex i to j at time t , which is updated each time the ants complete a cycle through the graph (every n iterations of the algorithm) according to the following formula:

$$\tau_{ij}(t+n) = \rho \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k$$

where ρ is a coefficient which determines how quickly pheromones evaporate along edges¹ and $\Delta\tau_{ij}^k$ is the amount per unit length of pheromone left on edge (i, j) by ant k between time t and $t+n$ which is given by

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ uses edge } (i, j) \text{ in its cycle} \\ 0 & \text{otherwise} \end{cases}$$

where Q is a constant and L_k is the cycle length of the ant. Note that “length” depends on what the application of ACO is, for example in the case of the Traveling Salesman Problem [11] we are simply considering the Euclidean distance between vertices, but in the DJSSP we would be considering time between vertices (jobs).

η_{ij} is a quantity known as the visibility from vertex i to vertex j . It is defined as $\frac{1}{d_{ij}}$, where d_{ij} is the Euclidean distance between vertex i and vertex j in the case of a routing problem like the TSP, but could be the time between job i and job j in the case of the DJSSP.

Finally, α and β are constants which determine the relative importance of pheromone strength and visibility when an ant has to choose to move to a different vertex. For example, setting $\alpha = 0$ will lead to pheromones being ignored and the result is a stochastic greedy algorithm. Optimal parameters are hard to determine for a particular problem, and are normally found experimentally [4].

Thus, as we can see, each ant chooses a new node based on a combination of pheromone strength and visibility. As the result is a probability, it is possible for ants to explore new paths even if a good path has already been found.

We show below a current working example on the Oliver30 nodes [2] where we use ACO to solve the TSP. We start off with our non-trivial nodes as shown in Figure 1 and the ants produce a closed path shown in Figure 2, which is very close to the known optimal (shortest) path.

¹Note that ρ must be less than 1 to prevent unlimited accumulation of pheromones.

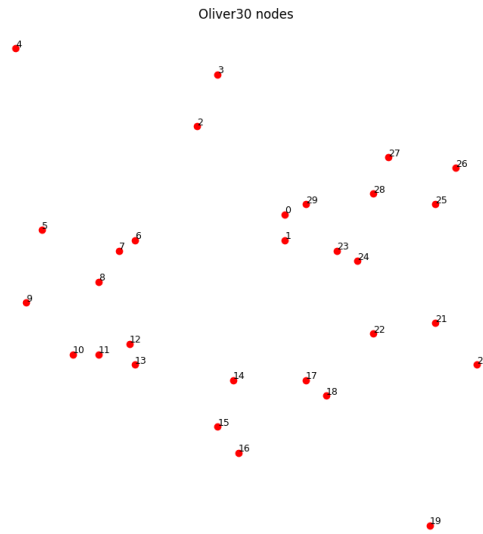


Figure 1: Oliver30 nodes

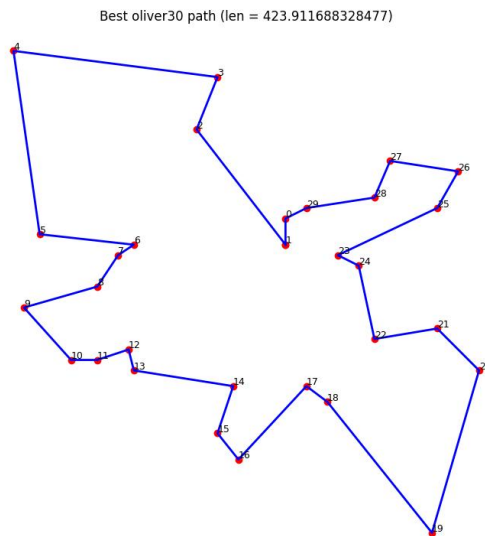


Figure 2: ACO-produced path on Oliver30

2.2 Dynamic Job Shop Scheduling Problem

The Dynamic Job Shop Scheduling Problem (DJSSP) extends the classic Job Shop Scheduling Problem (JSSP), whereby a set of jobs (a sequence of operations), must be scheduled on a finite set of machines. Each operation requires exclusive access to a specific machine for a known duration, and must be performed in a strict order. The goal is typically to minimise some metric of interest, such as makespan (total completion time), total tardiness, or machine idle time, while satisfying all operational constraints.

In the DJSSP, the scheduling environment can change over time. For example, new jobs may arrive unpredictably (e.g. emergency surgeries in a hospital), machines may break down, or previously scheduled operations may be delayed (e.g. an operation overruns). This introduces considerable complexity, as a previously valid schedule may become infeasible or suboptimal. The DJSSP thus requires rescheduling capabilities that are both fast and minimally disruptive.

2.3 ACO applied to DJSSP

Ant Colony Optimisation can be naturally extended to the DJSSP by interpreting the graph as a dynamic state space of partial schedules. Each vertex represents a possible operation assignment (job, machine, time), and ants construct feasible schedules by selecting valid next operations based on pheromone strength and heuristic information (operation priority, remaining slack time). When a disruption occurs, the affected corresponding operations are temporarily removed from the feasible set, and the ants go over the remaining schedule. Importantly, the pheromone trails left by previous solutions will be preserved, allowing ants to reuse previously computed good partial schedules and adapt quickly to the new constraints. This mechanism of distributed, memory-guided rescheduling makes ACO a promising method for solving DJSSP instances in environments where flexibility and reliability are essential, such as healthcare logistics.

3 Significance

There are many applications of DJSSP, spanning manufacturing, logistics [13] and healthcare [9]. In static environments, traditional solvers and heuristics such as branch and bound, tabu search, and genetic algorithms perform well [12]. However, most real-world applications tend to be dynamic, and we can never be certain if, in the context of a hospital, an emergency may arise or a ward might be full.

In certain scenarios, dynamic rescheduling after a disruption can unfortunately have life-changing consequences. A scheduling algorithm that is robust to change, adaptive to new constraints, and fast enough to produce feasible re-optimisations on the fly is of high practical value.

ACO should be well-suited to such dynamic scenarios, as it is fast and has natural memory in its pheromone trails. By building a Python package for solving the DJSSP (with an emphasis placed on faults, the goal is not to reinvent SLURM [6]), this project aims to fill a tooling gap in the applied scheduling community.

4 Review of Existing Work

As aforementioned, the static JSSP has been extensively studied [5][9][13], and there exist Python packages that solve it [7], usually using metaheuristic approaches like Genetic Algorithms. The DJSSP has been less thoroughly examined, although there are a few papers where ACO is used to solve DJSSP ([3][8]). However, these papers are proof of concept, with no open-source code or packages. To the best of our knowledge, there is no widely used, open-source Python package specifically targeting DJSSP with an ACO-based solver that can dynamically adjust schedules in real-time, which is where this project's novelty lies. Moreover, Genetic Algorithms are unsuitable for DJSSP as they need complete reinitialisation upon change, and Tabu search lacks memory reuse from past schedules, while ACO allows adaptive reuse of partial paths, making it better suited for dynamic scheduling environments.

5 Objectives

The IRP has the following objectives:

- **Develop a modular ACO-based Python package** specifically for DJSSP and make it publicly available.
- **Support a dynamic scheduling interface**, allowing insertion/removal of jobs or machines mid-simulation.
- **Model and simulate fault scenarios**, including emergency job arrivals, surgery overruns, and machine breakdowns.
- **Design benchmarks and experiments** comparing static ACO, naive rescheduling, and pheromone-guided rescheduling.
- **Apply the framework to a hospital use-case** with realistic constraints (multi-doctor, multi-room, emergency patients).

6 Future Plan

The project will span 12 weeks, divided into the following phases:

- **Weeks 1-2:** Literature review, definition of DJSSP constraints, and core design of ACO components (pheromone matrix, ant logic etc).
- **Weeks 3-5:** Develop static ACO scheduler. Validate on standard job shop benchmarks.
- **Weeks 6-7:** Extend to dynamic scheduling.
- **Weeks 8-9:** Implement fault models (e.g. emergency insertion, overruns) and test on synthetic hospital scenarios.
- **Weeks 10-11:** Run comparative experiments. Tune parameters and evaluate results. Investigate different ACO algorithms.
- **Week 12:** Finalise report, documentation, and testing.

7 Risks

Risk	Mitigation
ACO scalability under frequent disruptions	Implement time-bounded replanning
ACO scalability under large disruptions	Pheromone decay tuning to promote new schedules
Premature convergence	Tune parameters or try different ACO algorithms
Evaluating performance when disruptions are stochastic	Use a seed for repeatable testing

Table 1: Project risks and mitigation strategies

8 AI Acknowledgement Statement

Used ChatGPT-4o by OpenAI to help animate pheromone trails.

I confirm that all submitted work is my own, despite assistance received from genAI tools.

References

- [1] M. Dorigo, V. Maniezzo, and A. Coloni. “Ant system: optimization by a colony of cooperating agents”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.
- [2] Steve Dower. *Oliver 30 TSP*. URL: <https://stevedower.id.au/research/oliver-30> (visited on 06/11/2025).
- [3] Jeffrey Elcock and Nekiesha Edward. “An efficient ACO-based algorithm for task scheduling in heterogeneous multiprocessing environments”. In: *Array* 17 (2023), p. 100280. ISSN: 2590-0056. DOI: <https://doi.org/10.1016/j.array.2023.100280>. URL: <https://www.sciencedirect.com/science/article/pii/S259000562300005X>.
- [4] Dorian Gaertner and Keith Clark. “On Optimal Parameters for Ant Colony Optimization Algorithms.” In: *On Optimal Parameters for Ant Colony Optimization Algorithms*. Vol. 1. Jan. 2005, pp. 83–89.
- [5] D.J. Hootmt, P.B. Luh, and K.R. Pattipati. “A practical approach to job-shop scheduling problems”. In: *IEEE Transactions on Robotics and Automation* 9.1 (1993), pp. 1–13. DOI: 10.1109/70.210791.
- [6] M Jette and M Grondona. “SLURM: Simple Linux Utility for Resource Management”. In: (Dec. 2002). URL: <https://www.osti.gov/biblio/15002533>.
- [7] Pablo22. *JobShopLib: A JSSP Python package*. URL: https://github.com/Pablo22/job_shop_lib (visited on 06/11/2025).
- [8] Andrew Y.C. Nee Rong Zhou Heow Pueh Lee. “Applying Ant Colony Optimisation (ACO) algorithm to dynamic job shop scheduling problems”. In: *International Journal of Manufacturing Research* (2008). DOI: <https://doi.org/10.1504/IJMR.2008.019212>. URL: <https://www.inderscience.com/info/inarticle.php?artid=19212>.
- [9] Masoumeh Vali et al. “Application of job shop scheduling approach in green patient flow optimization using a hybrid swarm intelligence”. In: *Computers & Industrial Engineering* 172 (2022), p. 108603. ISSN: 0360-8352. DOI: <https://doi.org/10.1016/j.cie.2022.108603>. URL: <https://www.sciencedirect.com/science/article/pii/S0360835222005988>.
- [10] Wikipedia contributors. *Ant colony optimization algorithms — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-June-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Ant_colony_optimization_algorithms&oldid=1292514538.
- [11] Wikipedia contributors. *Travelling salesman problem — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-June-2025]. 2025. URL: https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1292600322.
- [12] Niels A. Wouda, Leon Lan, and Wouter Kool. “PyVRP: a high-performance VRP solver package”. In: *INFORMS Journal on Computing* 36.4 (2024), pp. 943–955. DOI: 10.1287/ijoc.2023.0055. URL: <https://doi.org/10.1287/ijoc.2023.0055>.
- [13] Longzhi Yang et al. “Job shop planning and scheduling for manufacturers with manual operations”. In: *Expert Systems* 38 (Aug. 2018), e12315. DOI: 10.1111/exsy.12315.